

1. Series and DataFrames

PanDas stands for "**Panel Data System**" and it's an open-source data analysis and manipulation tool.

It's built on top of **Numpy** library.

Two primary data structures used in Pandas,

1. Series
2. DataFrames

Series

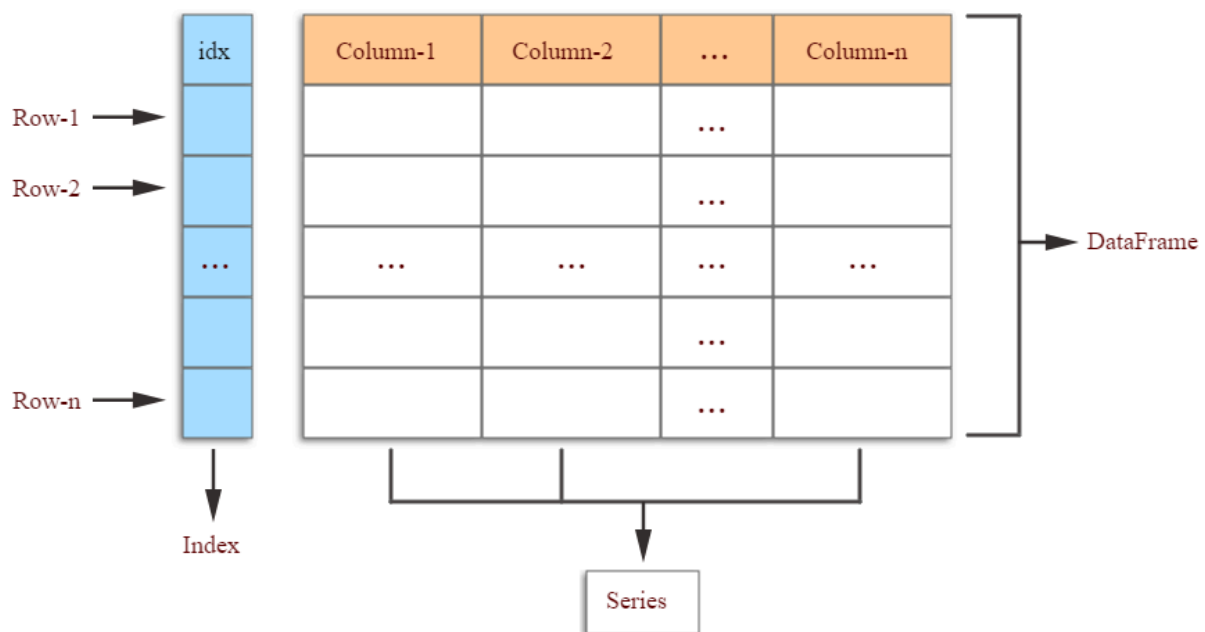
- A one-dimensional array like object(Imagine a vertical array).
- Able to hold any data type.
- Similar to a column in a Database table.
- Each element in a series is associated with an index(Used to uniquely identify elements). This index can be either a string or a number.

DataFrames

- Similar to a table in a relational database.
- A tabular data structure comprised of rows and columns.
- DataFrame can think of as a **group of Series** objects that share an index.

Series 1		Series 2		Series 3		Dataframe			
INDEX	DATA	INDEX	DATA	INDEX	DATA	INDEX	SERIES 1	SERIES 2	SERIES 3
0	A	0	1	0	[1, 2]	0	A	1	[1, 2]
1	B	1	2	1	A	1	B	2	A
2	C	2	3	2	1	2	C	3	1
3	D	3	4	3	(4, 5)	3	D	4	(4, 5)
4	E	4	5	4	{"a": 1}	4	E	5	{"a": 1}
5	F	5	6	5	6	5	F	6	6

Pandas Data structure



© w3resource.com

Creating Series and DataFrames :-

1. Create a Series from a list

```
main.py X
main.py > ...
1  import pandas as pd
2
3  data = [1, 2, 3, 4, 5]
4  series = pd.Series(data)
5  print(series)

PROBLEMS  OUTPUT  TERMINAL  ...  Python
● PS C:\Users\ranga\Desktop\python> python main.py
0    1
1    2
2    3
3    4
4    5
dtype: int64
○ PS C:\Users\ranga\Desktop\python> 
```

2. Creating a Series with custom index labels

```
main.py X
main.py > ...
1  import pandas as pd
2
3  data = [1, 2, 3, 4, 5]
4  idx = ['a', 'b', 'c', 'd', 'e']
5  series = pd.Series(data, index=idx)
6  print(series)

PROBLEMS  OUTPUT  TERMINAL  ...  Python
● PS C:\Users\ranga\Desktop\python> python main.py
a    1
b    2
c    3
d    4
e    5
dtype: int64
○ PS C:\Users\ranga\Desktop\python> 
```

3. Creating a DataFrame from a dictionary of lists

Python dictionary is a data structure that consist of key-value pairs.
Key can be used to identify each value and values can be any data type.

```
main.py X
main.py > ...
1 import pandas as pd
2
3 data = {
4     'A' : [1, 2, 3, 4],
5     'B' : [5, 6, 7, 8],
6     'C' : ['foo', 'bar', 'baz', 'sas']
7 }
8
9 df = pd.DataFrame(data)
10 print(df)
```

PROBLEMS OUTPUT TERMINAL ... Python

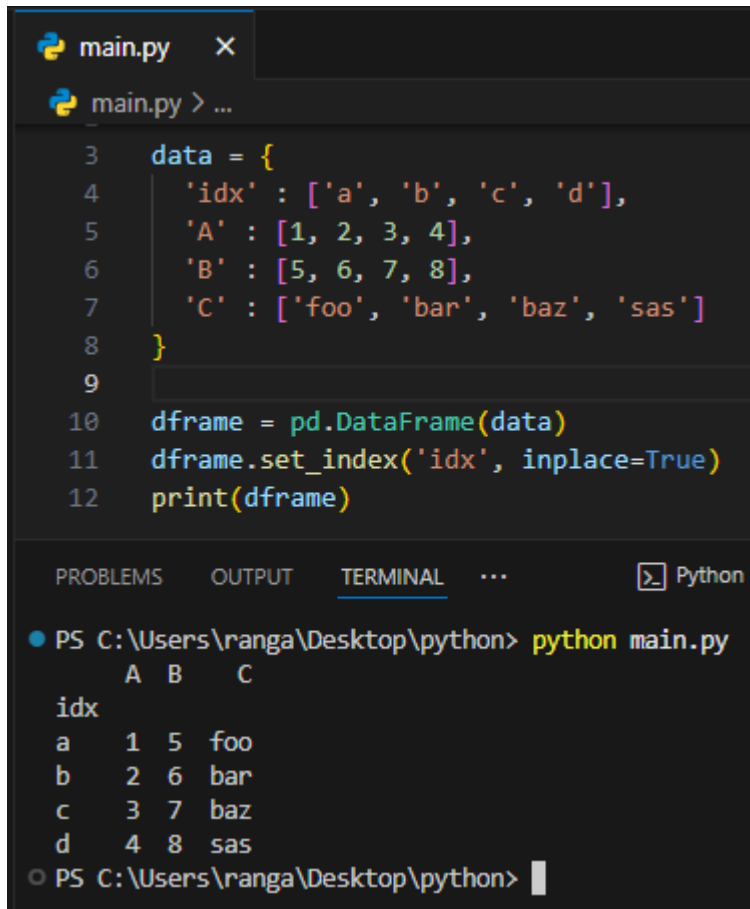
```
PS C:\Users\ranga\Desktop\python> python main.py
   A  B  C
0  1  5  foo
1  2  6  bar
2  3  7  baz
3  4  8  sas
PS C:\Users\ranga\Desktop\python> 
```

Here,

- All **A, B, C** key-value pairs were used as Series.
- Index was auto-generated. (0, 1, 2, 3)
- Keys(A, B, C) considered as column names.

We can tell panda to use one a key-value pair as the index of the DataFrame instead of

letting it generate automatically.



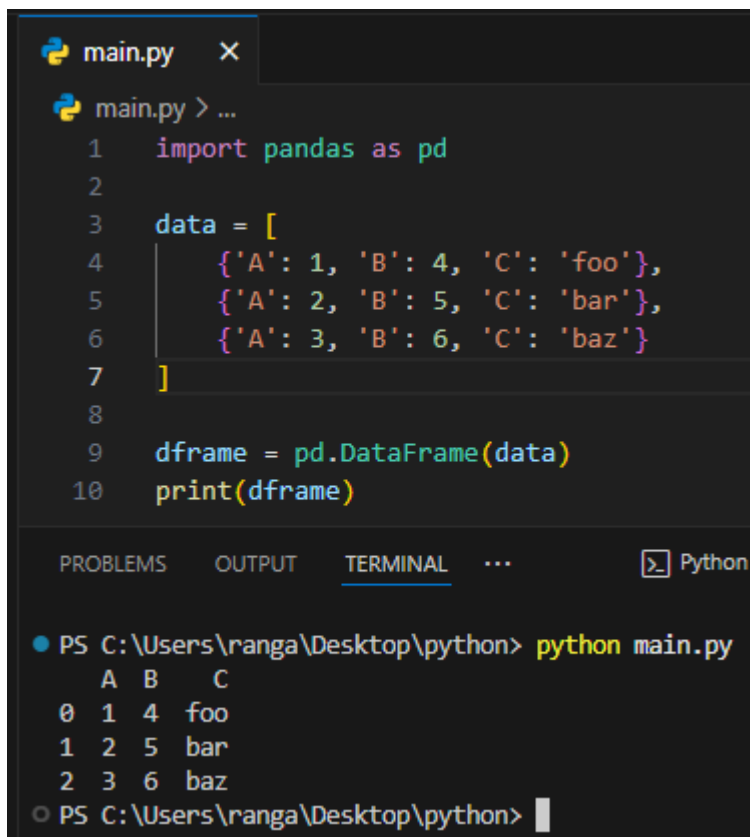
The screenshot shows a code editor with a file named `main.py`. The code defines a dictionary `data` with three keys: `'idx'`, `'A'`, and `'B'`. The values are lists of strings, integers, and integers respectively. A `pandas.DataFrame` is created from this dictionary, with `'idx'` set as the index. The terminal output shows the resulting DataFrame with columns `A`, `B`, and `C`, and index `idx`.

```
3 data = {
4     'idx' : ['a', 'b', 'c', 'd'],
5     'A' : [1, 2, 3, 4],
6     'B' : [5, 6, 7, 8],
7     'C' : ['foo', 'bar', 'baz', 'sas']
8 }
9
10 df = pd.DataFrame(data)
11 df.set_index('idx', inplace=True)
12 print(df)
```

Terminal output:

```
PS C:\Users\ranga\Desktop\python> python main.py
   A  B  C
idx
a   1  5 foo
b   2  6 bar
c   3  7 baz
d   4  8 sas
```

4. Creating a DataFrame from a list of dictionaries



The screenshot shows a code editor with a file named `main.py`. The code imports `pandas` as `pd` and defines a list of dictionaries `data`. Each dictionary has keys `'A'`, `'B'`, and `'C'` with integer and string values. A `pandas.DataFrame` is created from this list, and the terminal output shows the resulting DataFrame with columns `A`, `B`, and `C`, and index `0`, `1`, and `2`.

```
1 import pandas as pd
2
3 data = [
4     {'A': 1, 'B': 4, 'C': 'foo'},
5     {'A': 2, 'B': 5, 'C': 'bar'},
6     {'A': 3, 'B': 6, 'C': 'baz'}
7 ]
8
9 df = pd.DataFrame(data)
10 print(df)
```

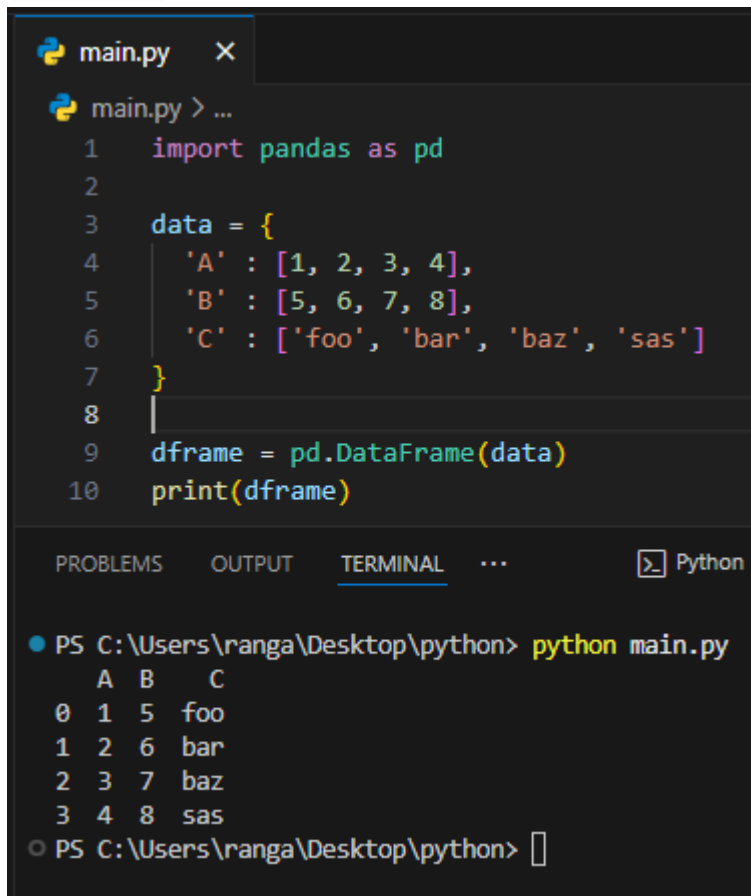
Terminal output:

```
PS C:\Users\ranga\Desktop\python> python main.py
   A  B  C
0   1  4 foo
1   2  5 bar
2   3  6 baz
```

Additional :-

1. `set_index` function

`set_index` function is used to set one or more columns of a DataFrame as the index.



The screenshot shows a Python IDE with a file named `main.py`. The code in the editor is as follows:

```
main.py > ...
1  import pandas as pd
2
3  data = {
4      'A' : [1, 2, 3, 4],
5      'B' : [5, 6, 7, 8],
6      'C' : ['foo', 'bar', 'baz', 'sas']
7  }
8
9  df = pd.DataFrame(data)
10 print(df)
```

The terminal output shows the execution of `python main.py`, resulting in a DataFrame with columns A, B, and C, and an auto-generated index (0, 1, 2, 3).

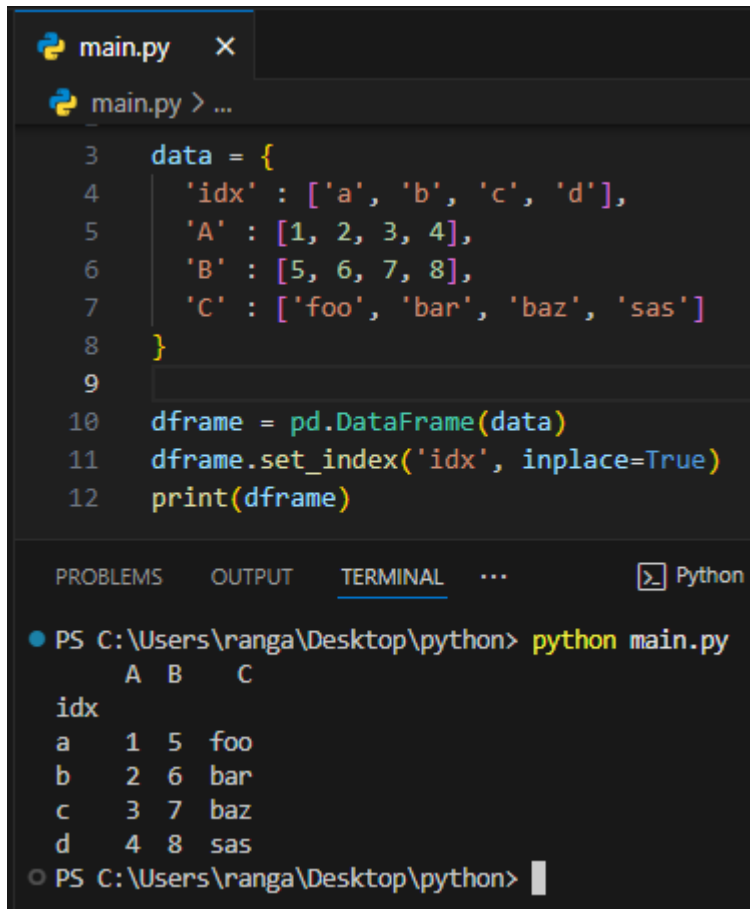
```
PS C:\Users\ranga\Desktop\python> python main.py
   A  B  C
0  1  5  foo
1  2  6  bar
2  3  7  baz
3  4  8  sas
PS C:\Users\ranga\Desktop\python> 
```

Here,

- All **A, B, C** key-value pairs were used as Series.
- Index was auto-generated. (0, 1, 2, 3)
- Keys(A, B, C) considered as column names.

We can tell panda to use one a key-value pair as the index of the DataFrame instead of

letting it generate automatically.



The screenshot shows a Python IDE with a file named `main.py`. The script defines a dictionary `data` with three keys: `'idx'`, `'A'`, and `'B'`. The `'idx'` key has a list of strings `['a', 'b', 'c', 'd']`. The `'A'` key has a list of integers `[1, 2, 3, 4]`. The `'B'` key has a list of integers `[5, 6, 7, 8]`. A `pd.DataFrame` is created from this dictionary, and the `'idx'` column is set as the index using `set_index('idx', inplace=True)`. The `df` is then printed.

```
3 data = {
4     'idx' : ['a', 'b', 'c', 'd'],
5     'A' : [1, 2, 3, 4],
6     'B' : [5, 6, 7, 8],
7     'C' : ['foo', 'bar', 'baz', 'sas']
8 }
9
10 df = pd.DataFrame(data)
11 df.set_index('idx', inplace=True)
12 print(df)
```

The terminal output shows the command `python main.py` being executed, resulting in the following DataFrame:

	A	B	C
idx			
a	1	5	foo
b	2	6	bar
c	3	7	baz
d	4	8	sas

2. `inplace` parameter

The `inplace` parameter is a Boolean flag that determines whether the operation is performed on the original DataFrame and then return the original DataFrame or the

modified DataFrame with changes.

```
main.py X
main.py > ...

2 data = {
3     'idx' : ['a', 'b', 'c', 'd'],
4     'A' : [1, 2, 3, 4],
5     'B' : [5, 6, 7, 8],
6     'C' : ['foo', 'bar', 'baz', 'sas']
7 }
8 dfT = pd.DataFrame(data)
9 dfF = pd.DataFrame(data)
10 dfT.set_index('idx', inplace=True)
11 dfF.set_index(['idx'])
12 print(dfT.index)
13 print(dfT)
14 print('')
15 print(dfF.index)
16 print(dfF)
```

PROBLEMS OUTPUT TERMINAL ... Python + v

```
PS C:\Users\ranga\Desktop\python> python main.py
Index(['a', 'b', 'c', 'd'], dtype='object', name='idx')
  A  B  C
idx
a   1  5 foo
b   2  6 bar
c   3  7 baz
d   4  8 sas

RangeIndex(start=0, stop=4, step=1)
  idx  A  B  C
0   a  1  5 foo
1   b  2  6 bar
2   c  3  7 baz
3   d  4  8 sas
```

- dfT DataFrame used `inplace=True` and its index is shown as `Index(['a', 'b', 'c', 'd'])`

When `inplace=True`, index setting function will work on the original DataFrame and will return a modified DataFrame with changes.

- dfF DataFrame used `inplace=False` (Default) and its index is shown as `RangeIndex(start=0, stop=4, step=1)`

When `inplace=False`, index setting function will work on the original DataFrame but it will return the original DataFrame without changes.